This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Currently Amended) A method for providing efficient access to an object in a computer system, the method comprising:

associating an identifier with an allocated object, wherein the identifier is associated with a first reuse count, ~~a second reuse count, and a validity indicator~~ and wherein the first reuse count is incremented when the identifier is used to access the object;

in response to a request to access the object: ~~associated with the identifier, using the validity indicator, the first reuse count, and the second reuse count to determine if the object is de-allocated~~;

comparing the first reuse count to a second reuse count, wherein the second reuse count is incremented when the identifier is used to access the object;

denying access to the object if the first reuse count does not match the second reuse count;

checking a validity indicator with at least two possible states comprising valid and invalid; and

denying access to the object if the validity indicator is in the invalid state.

~~if the object is de-allocated, denying access to the object.~~

2. (Original) The method of claim 1, wherein an entry associated with the identifier comprises a lock, the second reuse count, the validity indicator and an object reference.

3. (Original) The method of claim 2, wherein the entry is locked before accessing the entry and unlocked after accessing the entry.

4. (Original) The method of claim 2, further comprising in response to de-allocation of the object, setting the validity indicator associated with the identifier to indicate invalidity, and setting the object reference associated with the identifier to zero.

5. (Original) The method of claim 4, further comprising setting an object value associated with the object to zero.

6. (Original) The method of claim 4, further comprising in response to determining that second reuse count is less than a predetermined maximum value, placing the identifier on a list of available identifiers.

7. (Original) The method of claim 6, wherein the predetermined maximum value is determined by a size of a container in which the identifier is stored.

8. (Original) The method of claim 6, wherein the predetermined maximum value is greater than a total number of objects allocated between reboots.

9. (Original) The method of claim 1, wherein associating the object with the identifier further comprises:

retrieving the identifier from a list of available identifiers;

receiving a location in memory associated with the object;

setting the validity indictor associated with the identifier to indicate validity, setting an object reference associated with the identifier to the location in memory associated with the object, incrementing the second reuse count by a predefined value, setting the first reuse count to the second reuse count and setting an object value associated with the object to the identifier.

10. (Original) The method of claim 1, wherein determining that the object is de-allocated comprises at least one of:

determining that the first reuse count is not equal to the second reuse count; and

determining that the validity indicator indicates invalidity.

11. (Original) The method of claim 2, wherein the entry is stored in an array, wherein the array entry is indexed by the identifier.

12. (Canceled)

13. (Currently Amended) A method for managing an allocated object in a computer system, comprising:

providing a list of available ~~token~~ values;

providing a token comprising a ~~token data~~ value selected from the list and a first reuse count;

providing a ~~token~~ data array comprising an entry corresponding to the selected ~~token data~~ value, wherein the entry includes a validity indicator, the value of which is definable to indicate whether an object associated with the entry is valid and wherein the entry includes a second reuse count;

assigning the ~~token~~ value to the allocated object ~~by associating the token data value with the object~~;

in response to a request to perform an operation on the object, checking the validity indicator to determine whether the object is valid, determining if the first reuse count is equal to the second reuse count and permitting the operation to occur if the object is valid and the first reuse count is equal to the second reuse count.

14. (Original) The method of claim 13, wherein the entry also comprises an entry object reference, the value of which is definable to indicate where the object is located in memory.

15. (Original) The method of claim 14, further comprising:

receiving a location at which the allocated object resides, and

setting the entry object reference to the location at which the allocated object resides.

16. (Currently Amended) A method for managing a de-allocated object in a computer system, comprising:

providing a location in memory associated with a de-allocated object, the de-allocated object associated with a token data value;

determining the ~~token data~~ value associated with the de-allocated object;

providing a ~~token~~ data array comprising an entry corresponding to the ~~token data~~ value associated with the de-allocated object, wherein the entry includes a validity indicator, the value of which is definable to indicate whether an object associated with the entry is valid and wherein the entry includes an entry object reference, the value of which is definable to indicate where the object is located in memory;

in response to determining that the location in memory associated with the de-allocated object matches the entry object reference and to determining that the object is valid, setting the validity indicator to indicate invalidity, and setting the entry object reference to zero.

17. (Currently Amended) The method of claim 16, further comprising in response to determining that the ~~token data~~ value associated with the de-allocated object is less than a predefined value, adding the ~~token data~~ value to a list of available ~~token~~ values.

18. (Currently Amended) A computer system, comprising:

a processing unit; and

a memory including a data structure, the data structure comprising:

a list of available ~~token~~ values, ~~a token comprising a token data~~ wherein at least one value selected from the list ~~and~~ is associated with a first reuse count and wherein the first reuse count is incremented when the value is used to access an object;

~~and~~ a ~~token~~ data array comprising an entry corresponding to the selected ~~token data~~ value, wherein the entry includes:

a validity indicator the ~~value~~ state of which is definable to indicate whether an object associated with the entry is valid,

a second reuse count wherein the second reuse count is incremented when the value is used to access an object; and

an ~~entry~~ object reference.

19. (Currently Amended) The system of claim 18, wherein the ~~entry~~ object reference comprises a location in memory where ~~the~~ an object associated with the entry is located.

20. (Original) The system of claim 18, wherein the entry further includes a lock.

21. (Currently Amended) The system of claim 18, wherein the memory further includes ~~the~~ an object associated with the entry.

22. (Currently Amended) The system of claim 21, wherein the object associated with the entry is associated with the selected ~~token data~~ value by appending the value to the object associated with the entry.